

NoahFrame

——敏捷服务器开发解决方案介绍

NFrame Studio

NFrame

敏捷服务器开发解决方案

什么是NoahFrame?

NoahFrame(以下**简称NFrame**)是一个使用C++语言开发的、支持高并发、高性能的跨平台敏捷服务器开发解决方案。旨在帮助中小企业降低开发门槛，快速完成项目功能。

采用敏捷开发中的分层设计思路，将功能拆分为多个插件模块，让开发人员集中处理单一功能，提高团队效率。

NFrame经历过国内最知名游戏研发运营厂商的考验，设计了一套可动态扩展的服务器架构和逻辑架构，可节约底层约70%的代码量、节约架构层约90%的代码量、节约逻辑层约60%代码量，让研发团队专注于产品设计，不再考虑较多技术问题，大幅节约项目时间。

NFrame为全平台服务器应用而生，支持开发PC、手机端服务，更加适合现在流行的移动互联网服务。

NFrame良好的设计可以使其轻松适配多种类型的开发需求，无论是端游、页游、手游、甚至工业应用均可使用NF进行开发。

NFrame

敏捷服务器开发解决方案

NFrame的特性

- 通用的抽象对象系统
- 数据驱动 (Property & record)
- 事件驱动 (Event)
- 可扩展的App、插件化、模块化 (Plugin & Module)
- 面向接口编程 (IOD)
- 高性能、高并发 (网络、Actor、逻辑)
- Component组件 (脚本系统)
- 分布式服务器架构
- 高稳定性、简易部署、支持扩展、跨平台
- 可视化配置、配套工具
- 配套客户端(Unity3D客户端、Cocos2D客户端)
- 企业定制化服务(存储方案、逻辑模块、新架构)

NFrame

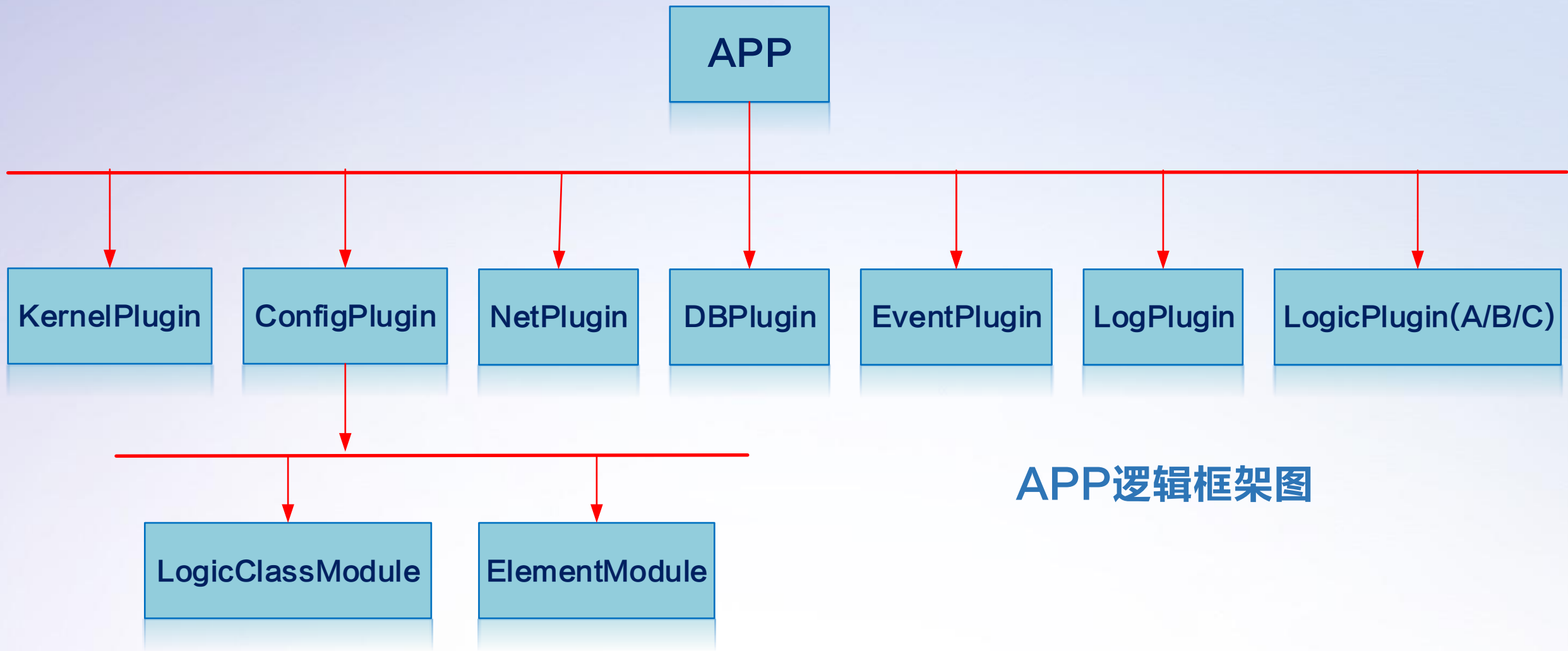
敏捷服务器开发解决方案

通用的抽象对象系统(LogicClass)

- 丰富的基础属性类型
- 对象属性的可配置性(XML可以定义所有属性)
- 对象初始数据的可配置性(XML可以预设值所有属性的值)
- 可动态增减属性(服务器运行过程中可以程序添加属性)
- 无需在代码中再声明任何业务类(XML直接声明)
- 通用的设置/获取信息接口

NFrame

敏捷服务器开发解决方案



APP逻辑框架图

NFrame

敏捷服务器开发解决方案

LogicClass文件示例

```
<?xml version="1.0" encoding="utf-8"?>
<!--this is logic class xml-->
<XML>
  <Class Desc="IObject" Id="IObject" InstancePath="../../NFDataCfg/Ini/NPC/IObject.xml" Path="../../NFDataCfg/Struct/Class
    <Class Desc="BB_Build" Id="BB_Build" InstancePath="../../NFDataCfg/Ini/NPC/BB_Build.xml" Path="../../NFDataCfg/Struc
    <Class Desc="BB_Player" Id="BB_Player" InstancePath="../../NFDataCfg/Ini/NPC/BB_Player.xml" Path="../../NFDataCfg/St
    <Class Desc="Buff" Id="Buff" InstancePath="../../NFDataCfg/Ini/NPC/Buff.xml" Path="../../NFDataCfg/Struct/Class/Buff
    <Class Desc="EffectData" Id="EffectData" InstancePath="../../NFDataCfg/Ini/NPC/EffectData.xml" Path="../../NFDataCfg
    <Class Desc="Equip" Id="Equip" InstancePath="../../NFDataCfg/Ini/NPC/Equip.xml" Path="../../NFDataCfg/Struct/Class/E
    <Class Desc="GameServer" Id="GameServer" InstancePath="../../NFDataCfg/Ini/NPC/GameServer.xml" Path="../../NFDataCfg
    <Class Desc="InitProperty" Id="InitProperty" InstancePath="../../NFDataCfg/Ini/NPC/InitProperty.xml" Path="../../NFD
    <Class Desc="Item" Id="Item" InstancePath="../../NFDataCfg/Ini/NPC/Item.xml" Path="../../NFDataCfg/Struct/Class/Item
    <Class Desc="LoginServer" Id="LoginServer" InstancePath="../../NFDataCfg/Ini/NPC/LoginServer.xml" Path="../../NFData
    <Class Desc="MasterServer" Id="MasterServer" InstancePath="../../NFDataCfg/Ini/NPC/MasterServer.xml" Path="../../NFD
    <Class Desc="NPC" Id="NPC" InstancePath="../../NFDataCfg/Ini/NPC/NPC.xml" Path="../../NFDataCfg/Struct/Class/NPC.xml
    <Class Desc="Player" Id="Player" InstancePath="../../NFDataCfg/Ini/NPC/Player.xml" Path="../../NFDataCfg/Struct/Clas
    <Class Desc="ProxyServer" Id="ProxyServer" InstancePath="../../NFDataCfg/Ini/NPC/ProxyServer.xml" Path="../../NFData
    <Class Desc="Scene" Id="Scene" InstancePath="../../NFDataCfg/Ini/NPC/Scene.xml" Path="../../NFDataCfg/Struct/Class/S
    <Class Desc="Shop" Id="Shop" InstancePath="../../NFDataCfg/Ini/NPC/Shop.xml" Path="../../NFDataCfg/Struct/Class/Sho
    <Class Desc="Skill" Id="Skill" InstancePath="../../NFDataCfg/Ini/NPC/Skill.xml" Path="../../NFDataCfg/Struct/Class/S
    <Class Desc="SqlServer" Id="SqlServer" InstancePath="../../NFDataCfg/Ini/NPC/SqlServer.xml" Path="../../NFDataCfg/St
    <Class Desc="StateFuncResources" Id="StateFuncResources" InstancePath="../../NFDataCfg/Ini/NPC/StateFuncResources.xml
    <Class Desc="StateFunction" Id="StateFunction" InstancePath="../../NFDataCfg/Ini/NPC/StateFunction.xml" Path="../../
    <Class Desc="Task" Id="Task" InstancePath="../../NFDataCfg/Ini/NPC/Task.xml" Path="../../NFDataCfg/Struct/Class/Task
    <Class Desc="WorldServer" Id="WorldServer" InstancePath="../../NFDataCfg/Ini/NPC/WorldServer.xml" Path="../../NFData
  </Class>
</XML>
```

NFrame

敏捷服务器开发解决方案

数据驱动 (Property & record)

相对于传统的服务器开发，NFrame使用了一种全新的数据定义和使用的方法，我们称之为**属性(Property)**和**表(Record)**。

属性(Property)主要用来存储用户的基本数据，例如：姓名、性别、年龄、等级 等数据，主要表现为一个名称对应一个数据。

表(Record)主要用来存储一些记录，例如：道具列表、任务列表 等数据，主要表现为一个记录里包含多条数据。

NFrame使用了此种模型来定义应用中的所有数据，**避免了**以往传统服务器中**数据结构定义混乱，接口不统一、别人无法接手**等问题。

该功能由上图中提到的PropertyManager和RecordManager来管理。

NFrame

敏捷服务器开发解决方案

一个数据定义的例子：

```
<Proper<Record Col="18" Desc="普通背包" Id="BagItemList" Index="0" Private="1" Public="0" RelationValue="Friend" Row="211" Save="1"
  <Pr    <Col Tag="GUID" Type="object"/>                                :ring" V
  <Pr    <Col Tag="ConfigID" Type="string"/>                            r="1"/>
  <Pr    <Col Tag="ItemCount" Type="int"/>                                :w="1"/>
  <Pr    <Col Tag="Bound" Type="int"/>                                    :w="1"/>
  <Pr    <Col Tag="ExpiredType" Type="int"/>                            .end" Sa
  <Pr    <Col Tag="SlotCount" Type="int"/>                                ' Public
  <Pr    <Col Tag="InlayStone1" Type="string"/>                        e="strin
  <Pr    <Col Tag="InlayStone2" Type="string"/>                        e="strin
  <Pr    <Col Tag="InlayStone3" Type="string"/>                        e="obje
  <Pr    <Col Tag="InlayStone4" Type="string"/>                        ve="1"
  <Pr    <Col Tag="InlayStone5" Type="string"/>                        SaveInt
  <Pr    <Col Tag="InlayStone6" Type="string"/>                        'int" Vi
  <Pr    <Col Tag="EnhancedLevel" Type="string"/>                      r="1"/>
  <Pr    <Col Tag="SagecraftLevel" Type="int"/>                        aveInte
  <Pr    <Col Tag="SagecraftStone" Type="string"/>                    :w="1"/>
  <Pr    <Col Tag="RandProperty" Type="int"/>                          :="1"/>
  <Pr    <Col Tag="Date" Type="object"/>                                :w="1"/>
  <Pr    <Col Tag="BaseProperty" Type="string"/>                      ="1"/>
  <Pr</Record>                                                         ="string
```

NFrame

敏捷服务器开发解决方案

事件驱动(EventDriver)

事件驱动灵感来源与处理器的处理流程，旨为只提供流水线式的处理逻辑模块，而本身不保存和涉留对象的数据

事件驱动包含：Property驱动，Record驱动，Event驱动，Heartbeat驱动

通过Property Driver，所有只要注册过属性观测者的Processor Function均会得到所关注Property的变化通知，以便做出对应的逻辑处理。

通过Record Driver，所有只要注册过Record的观测者的Processor Function均会得到所关注Record的变化通知，以便做出对应的逻辑处理。

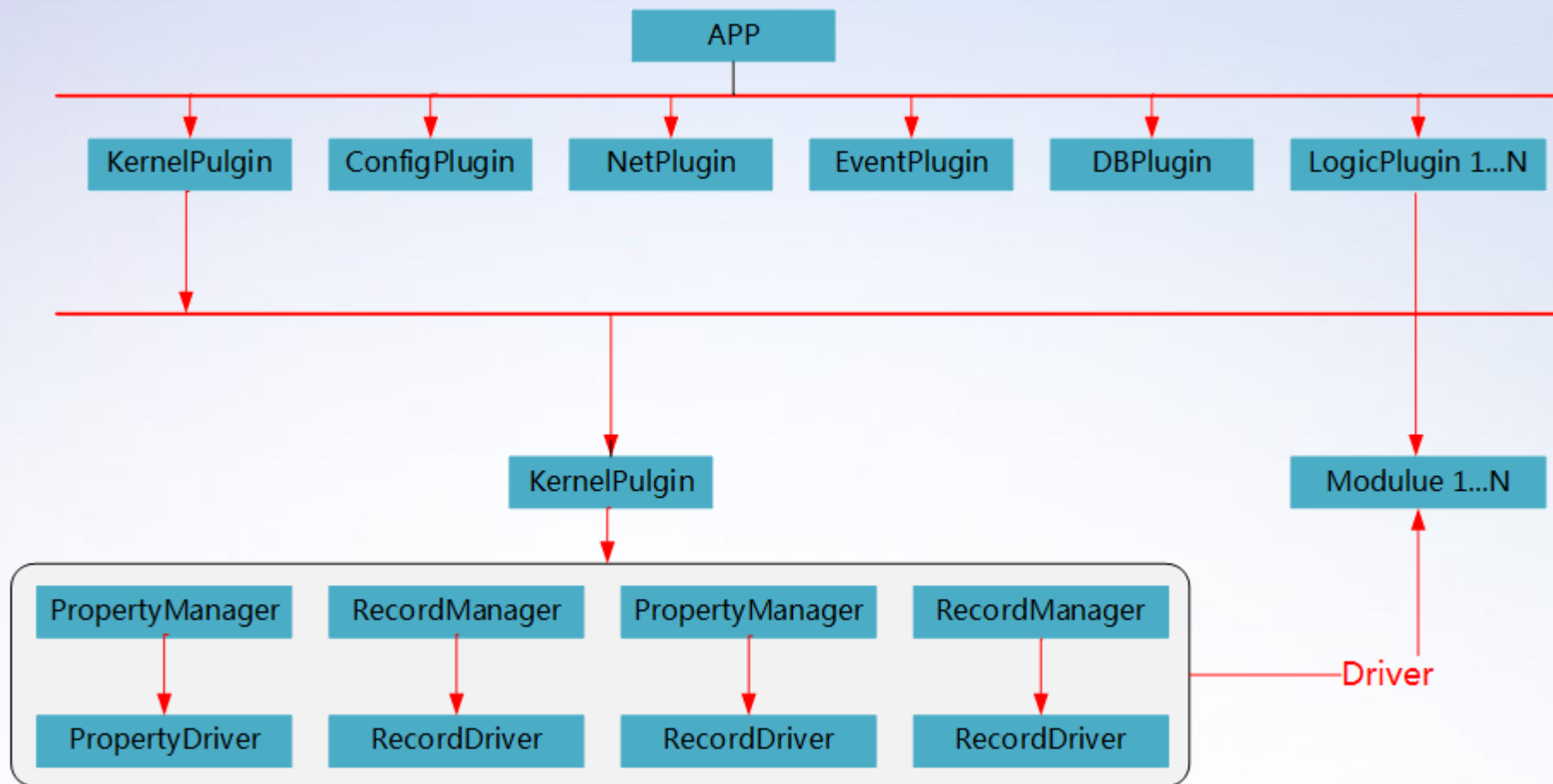
通过Event System，所有只要注册过Event的观测者的Processor Function均会得到所关注的事件通知Processor可以产生新的事件或属性驱动，以便驱动其他逻辑模块处理逻辑。

通过Heartbeat System，所有只要注册过同名心跳的观测者的Processor Function均会定时处理逻辑，以便延时/定时处理逻辑。

NFrame

敏捷服务器开发解决方案

事件驱动设计图



NFrame

敏捷服务器开发解决方案

Property驱动

```
m_pKernelModule->AddPropertyCallBack(self, "Level", this, &NFCPropertyModule::OnObjectLevelEvent);

m_pKernelModule->SetPropertyInt(self, "Level", 100);

int NFCPropertyModule::OnObjectLevelEvent( const NFIDENTID& self, const std::string& strPropertyName,
    const NFIDataList& oldVar, const NFIDataList& newVar, const NFIDataList& argVar )
{
    // do something
    return 0;
}
```

Record驱动

```
m_pKernelModule->AddRecordCallBack(self, "PropertyRecord", this, &NFCPropertyModule::OnRecordPropertyEvent);

int NFCPropertyModule::OnRecordPropertyEvent( const NFIDENTID& self, const std::string& strRecordName,
    const int nOpType, const int nRow, const int nCol, const NFIDataList& oldVar,
    const NFIDataList& newVar, const NFIDataList& argVar )
{
    // do something

    return 0;
}
```

Event驱动

```
m_pEventProcessModule->AddEventCallback( self, NFED_ON_CLIENT_ENTER_SCENE, this, &NFCSceProcessModule::OnEnterSceneEvent );

NFCDatList varEntry;
varEntry << pObj->Self();
varEntry << 0;
varEntry << nSceneID;
varEntry << -1;
m_pEventProcessModule->DoEvent( pObj->Self(), NFED_ON_CLIENT_ENTER_SCENE, varEntry );

int NFCSceProcessModule::OnEnterSceneEvent( const NFIDENTID& self, const int nEventID, const NFIDatList& var )
{
    // do something
    return 0;
}
```

Heartbeat驱动

```
m_pKernelModule->AddHeartBeat(self, "OnHeartBeat", this, &HelloWorld3Module::OnHeartBeat, NFCDatList(), 5.0f, 9999 );

int HelloWorld3Module::OnHeartBeat(const NFIDENTID& self, const std::string& strHeartBeat,
    const float fTime, const int nCount, const NFIDatList& arg)
{
    // do something
    return 0;
}
```

NFrame

敏捷服务器开发解决方案

可扩展的App、插件化、模块化

NFrame采用国外成熟使用的敏捷开发思想——**分层设计**。

分层的程序设计带来的好处是显而易见的，由于层间松散的耦合关系，使得我们可以专注于本层的设计，而不必关心其他层的设计，也不必担心自己的设计会影响其它层，对提高软件质量大有裨益。

而且分层设计使得程序结构清晰，升级和维护都变得十分容易，更改层的具体实现代码，只要层接口保持稳定，其他层可以不必修改。即使层的接口发生变化，也只影响上层和下层，修改工作量小而且错误可以控制，不会带来意外的风险。

NFrame同时使用了将应用程序设计成三层架构，最顶层是App，中间层是各种插件，插件下是各种对应的具化的模块功能。这种设计的优点是对应模块只处理自己的事务，降低耦合，通过接口与其他模块交互，将模块的风险降到最低。

NFrame

敏捷服务器开发解决方案

面向接口编程(IOD)

较于大多数OO式开发，NFrame支持更灵活的IO(接口)式开发，让你的开发更简单纯粹。通过模块抽象基类的虚接口让模块的功能互相调用，真正做到了软件开发的低耦合高内聚。

```
#include "NFILogicModule.h"

class NFIPropertyModule
    : public NFILogicModule
{
public:

    virtual int RefreshBaseProperty(const NFIDENTID& self) = 0;

    virtual bool FullHPMP(const NFIDENTID& self) = 0;
    virtual bool AddHP(const NFIDENTID& self, int nValue) = 0;
    virtual bool ConsumeHP(const NFIDENTID& self, int nValue) = 0;

    virtual bool AddMP(const NFIDENTID& self, int nValue) = 0;
    virtual bool ConsumeMP(const NFIDENTID& self, int nValue) = 0;

    virtual bool ConsumeSP(const NFIDENTID& self, int nValue) = 0;
    virtual bool FullSP(const NFIDENTID& self) = 0;
    virtual bool AddSP(const NFIDENTID& self, int nValue) = 0;
```

NFrame

敏捷服务器开发解决方案

高性能、高并发

NFrame由于设计上的分层独立从而使得架构上本身就性能较高。同时在网络通信上使用了久经考验的LibEvent作为网络底层，使用google ProtoBuf作为协议序列化库，LibEvent的高性能加上Protobuf的高压缩率，真实测试过单服承载8000以上用户高频率协议通讯。

NFrame在逻辑处理上使用了Actor模式，采用了Theron作为Actor基础类库，支持并发编程，充分利用CPU性能。Theron作为知名工业级并发库，应用在许多工业级软件上，例如Matlab使用了Theron后，让其性能直接提高了4.5x倍([原文链接](#))。

NFrame使用C++作为基础开发语言，相对于其他编程语言，在性能和效率上更是快人一步，良好的设计模式的应用让逻辑更加简单。

NFrame

敏捷服务器开发解决方案

Component组件

- 提供类似Unity样式的组件组合模式，以丰富服务器后期脚本编辑以及对象行为扩展
- 系统在对象产生时自动识别以及实例化Component
- 通过事件系统，向Component注入事件消息，便于处理逻辑
- 特殊需求的特殊处理，灵活拓展对象的行为
- 不用关心其他模块，降低耦合
- 当前已经支持lua作为脚本嵌入到框架中使用(详见教程6)
- 后期将支持JavaScript，C#，python等脚本语言

NFrame

敏捷服务器开发解决方案

分布式服务器架构

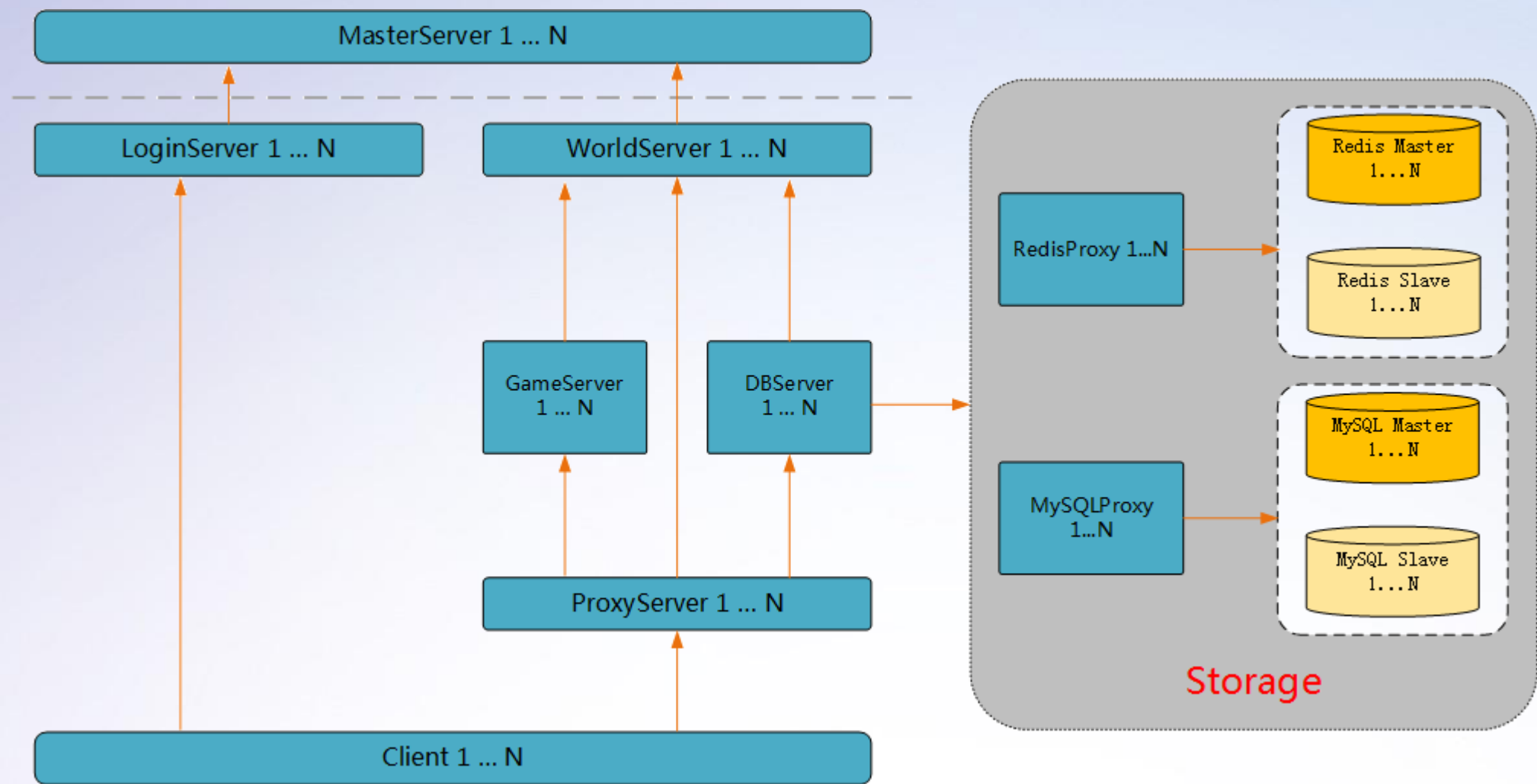
经过我们参考主流架构以及长期的项目经验积累，NFrame被设计为理论上可无限扩展的分布式架构，拥有着真正的无理论上限的系统规模和负载可扩展性。

主要特性：

- ✓ 无限扩展
- ✓ 高可靠性
- ✓ 简易通讯
- ✓ 业务去中心化
- ✓ 自动摘除问题节点
- ✓ 节点实时权重运算和评估
- ✓ 负载均衡

NFrame

敏捷服务器开发解决方案



NFrame服务器架构图

良好架构带来的好处

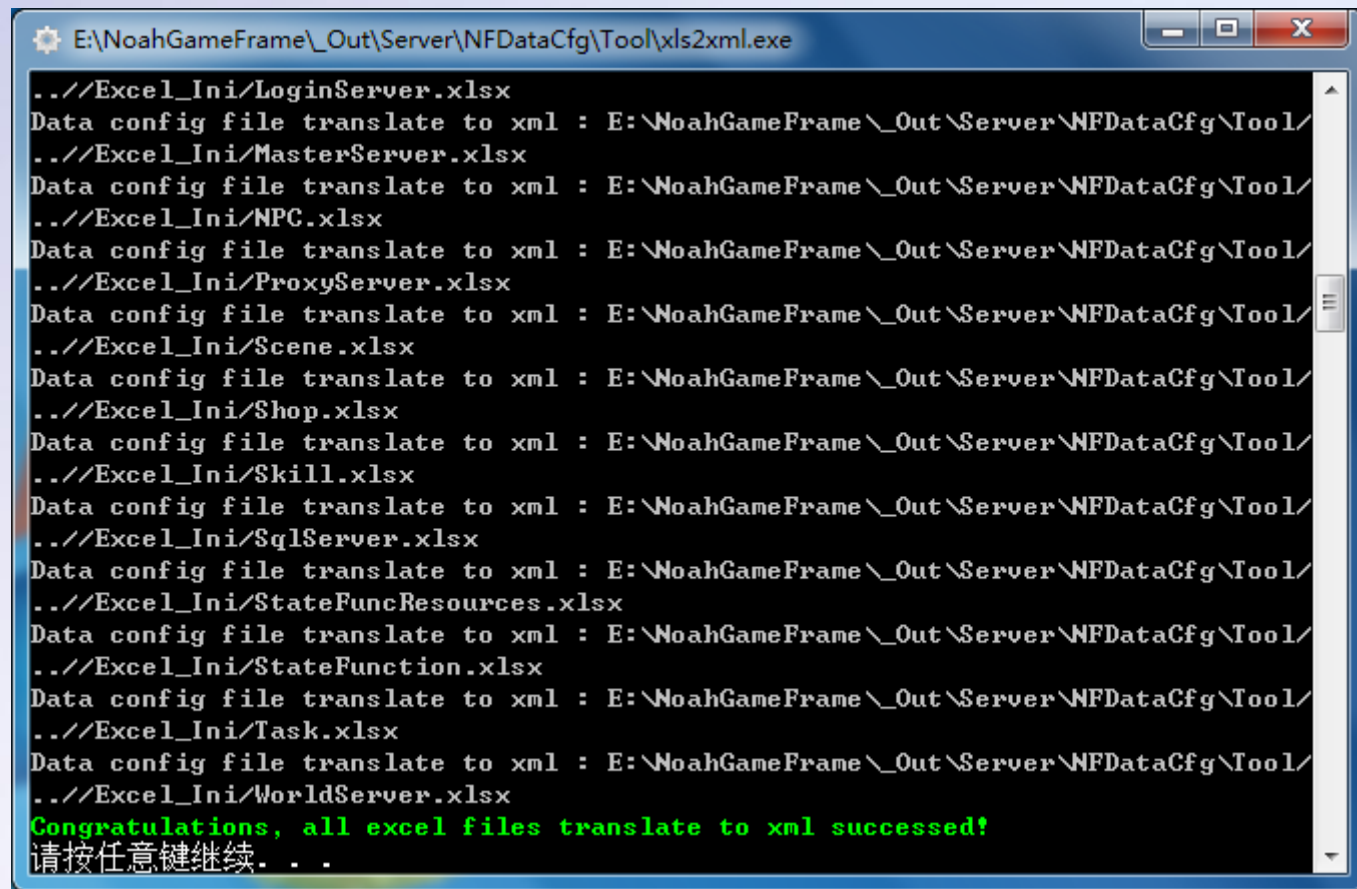
由于NFrame良好的服务器架构设计，故而拥有了其他架构所没有的优点。

- ✓ **高性能**，通讯优化，压缩流量，C++底层，业务逻辑并行
- ✓ **高稳定性**，服务器间实时数据统计，问题节点自动摘除，业务逻辑不中断
- ✓ **简易部署**，通过开服工具可自动部署配置文件
- ✓ **方便扩展**，实时热加载配置，横向无限扩展
- ✓ **跨平台**，使用标准C++开发，Windows和Linux均可使用，屏蔽大多数系统差异
- ✓ **节约硬件成本**，分布式架构，运营厂商的每一台机器都可作为节点加入到业务网络中，性能低的机器照样可以使用
- ✓ **维护简单**，实时通过log和分析工具查看服务器状态以及负载

NFrame

敏捷服务器开发解决方案

可视化配置、配套工具



```
E:\NoahGameFrame\Out\Server\NFDataCfg\Tool\xls2xml.exe

../Excel_Ini/LoginServer.xlsx
Data config file translate to xml : E:\NoahGameFrame\Out\Server\NFDataCfg\Tool\
../Excel_Ini/MasterServer.xlsx
Data config file translate to xml : E:\NoahGameFrame\Out\Server\NFDataCfg\Tool\
../Excel_Ini/NPC.xlsx
Data config file translate to xml : E:\NoahGameFrame\Out\Server\NFDataCfg\Tool\
../Excel_Ini/ProxyServer.xlsx
Data config file translate to xml : E:\NoahGameFrame\Out\Server\NFDataCfg\Tool\
../Excel_Ini/Scene.xlsx
Data config file translate to xml : E:\NoahGameFrame\Out\Server\NFDataCfg\Tool\
../Excel_Ini/Shop.xlsx
Data config file translate to xml : E:\NoahGameFrame\Out\Server\NFDataCfg\Tool\
../Excel_Ini/Skill.xlsx
Data config file translate to xml : E:\NoahGameFrame\Out\Server\NFDataCfg\Tool\
../Excel_Ini/SqlServer.xlsx
Data config file translate to xml : E:\NoahGameFrame\Out\Server\NFDataCfg\Tool\
../Excel_Ini/StateFuncResources.xlsx
Data config file translate to xml : E:\NoahGameFrame\Out\Server\NFDataCfg\Tool\
../Excel_Ini/StateFunction.xlsx
Data config file translate to xml : E:\NoahGameFrame\Out\Server\NFDataCfg\Tool\
../Excel_Ini/Task.xlsx
Data config file translate to xml : E:\NoahGameFrame\Out\Server\NFDataCfg\Tool\
../Excel_Ini/WorldServer.xlsx
Congratulations, all excel files translate to xml succeeded!
请按任意键继续...
```

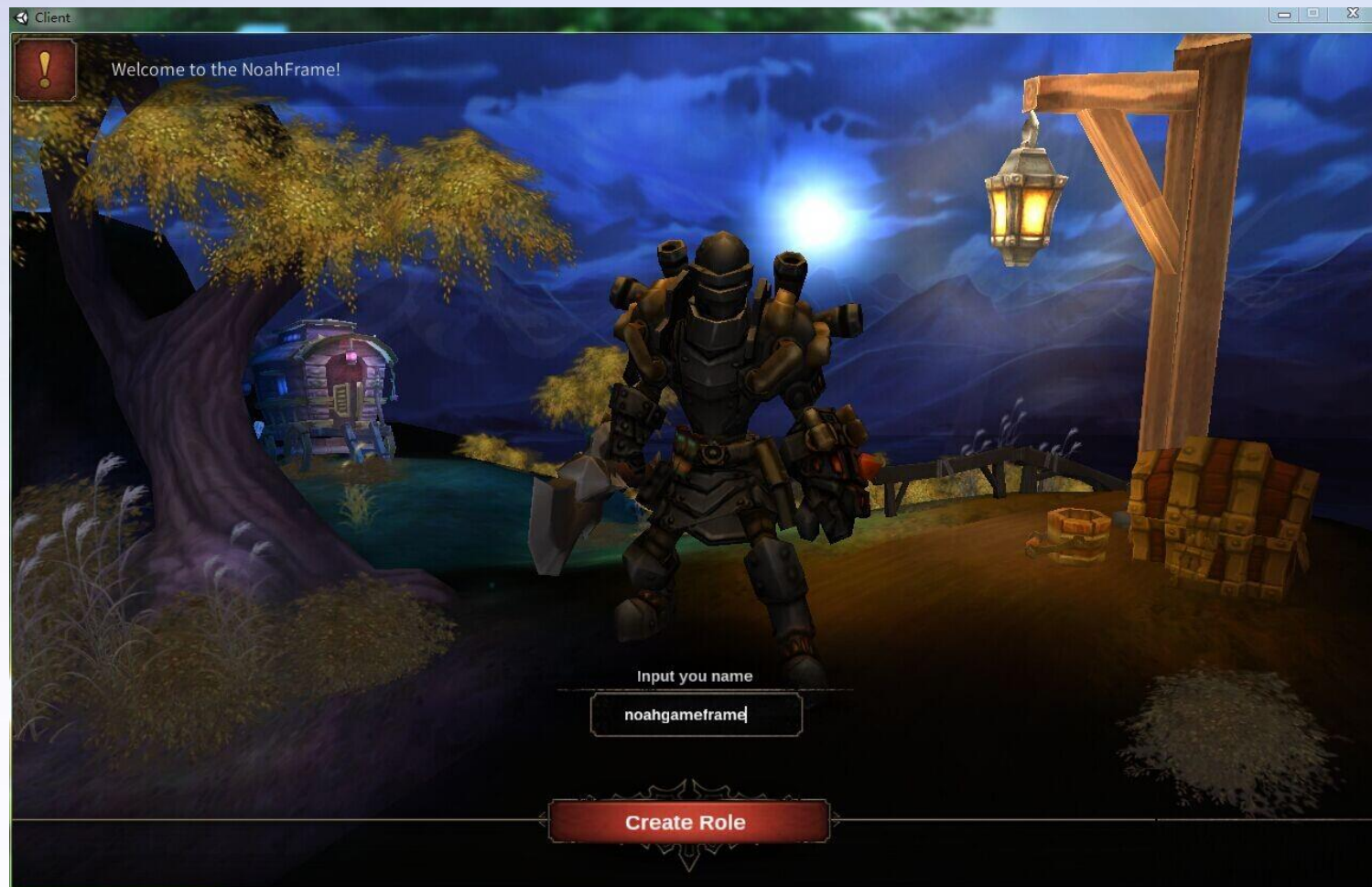
为方便与非编程人员合作，NFrame支持使用广泛使用的**Office Excel**进行数据和资源的配置，让合作部门能更快并准确的进行配置，降低部门合作的门槛。

同时我们提供**Excel转化成xml的工具**，方便程序内部使用，降低程序读取配置的难度。

NFrame

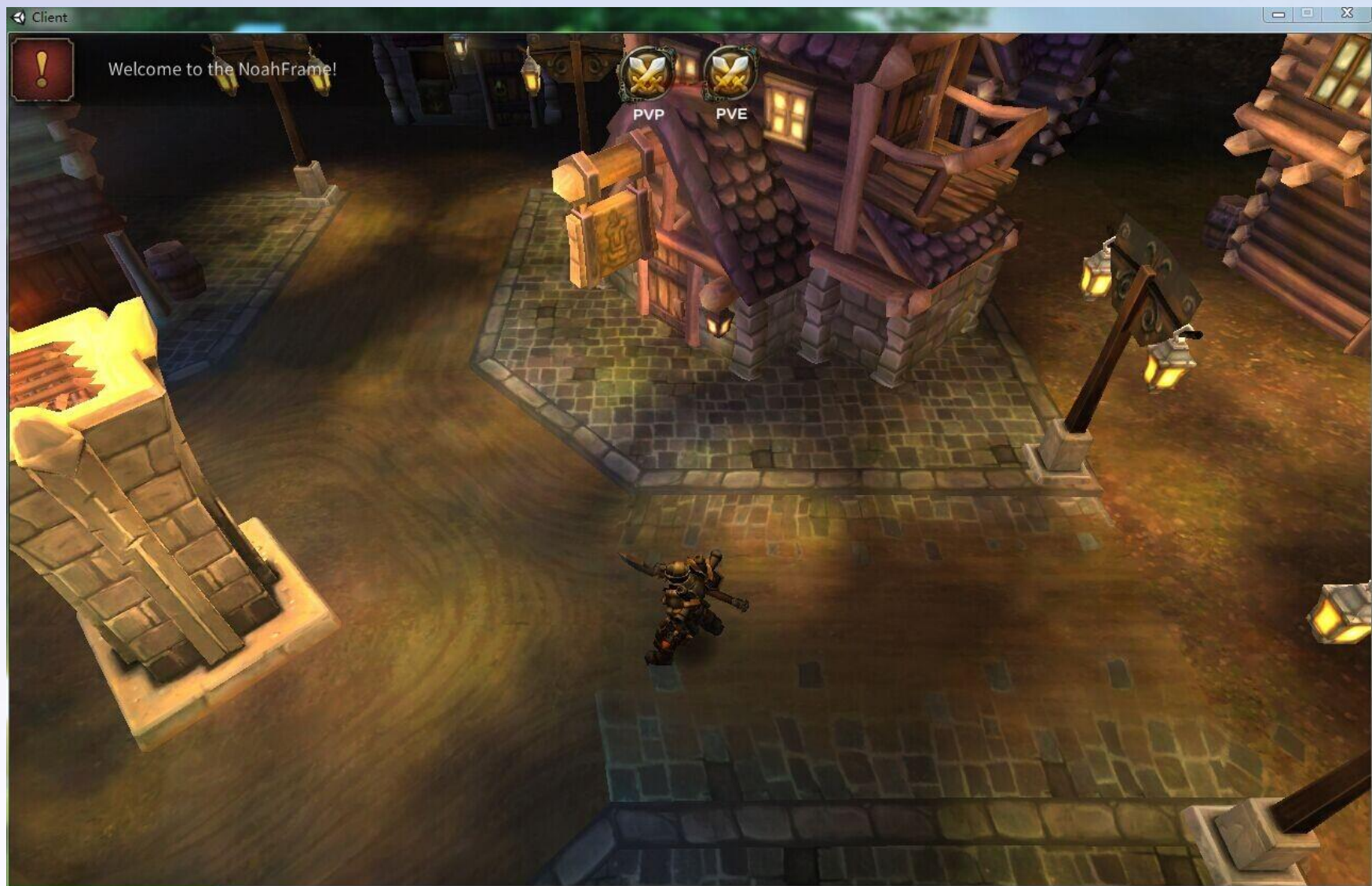
敏捷服务器开发解决方案

配套客户端



NFrame

敏捷服务器开发解决方案



NFrame

敏捷服务器开发解决方案

企业定制化服务

我们提供整套的解决方案，但是考虑到部分企业的业务需求与我们设计的有一定差别，所以我们面向企业提供特别定制化方案。

- ✓ 企业特定业务服务器架构
- ✓ 企业特定逻辑模块
- ✓ 企业特定网络通信组件
- ✓ 企业特定语言嵌入式组件
- ✓ 企业各种业务存储方案
- ✓ 企业特定项目整体设计开发
- ✓ 企业特定运营硬件环境
- ✓ 企业其他业务需求

NFrame

敏捷服务器开发解决方案

后期开发计划

- 完善Unity3D ARPG客户端(逻辑，表现等)
- 开发Cocos2D客户端
- 支持更多脚本语言(JavaScript, C#)
- 服务器架构、性能、功能持续优化
- 服务器更多模块组件支持(排行榜，PVP，PVE，AI，强互动等)
- Web运维工具
- Web日志分析
- 游戏SaaS云
- More...

NFrame

敏捷服务器开发解决方案

One more thing

为什么要选择NFrame?

NFrame

敏捷服务器开发解决方案

NFrame如何为企业节约成本？

作为企业，最关心的自然是性价比，如何将成本最大化是一个最大的问题？

一个小型的游戏**研发**团队主要成员大致如下图(注意：这里并未计算场地/水电/桌椅/计算机等基础设施)

职位	人数(人)	月薪(元，以上海为例)
制作人/负责人	1	20000
美术	2	10000
策划	2	10000
服务器程序	1	15000
客户端程序	1--2	15000
	7--8	90000--105000
		平均值 100000

研发周期最少6个月，多则12个月，下来进入运营阶段，工资还是得照常发放。算下来需要150万——300万的研发费用，这里是按照最少人员配置来计算，并且每个人都可以胜任自己的岗位。如果团队人员出现流动的情况，研发周期和费用会更大。

NFrame可以做到一个月出demo，让项目快速进入风投和天使阶段，同时我们提供的定制服务，最大化为企业节约时间和金钱。

NFrame

敏捷服务器开发解决方案

与其他传统框架的对比

对比项	NFrame	传统服务器框架
开放性	服务器架构代码完全开放	部分代码缺失
引擎质量	团队整体开发，质量较高	流出自某些公司，人员参差不齐，质量良莠不齐
架构质量	最新的分布式架构，无理论承载上限	传统端游小服小区架构，无法承载较高用户
完整性	全套完整框架和逻辑	部分核心模块缺失
开发难度	简单上手，复杂模块基本自带	难于上手，熟悉代码时间较长
人员要求	普通逻辑程序员即可开发	资深高级程序员
开发时间	一个月出可玩demo	3-5个月内出demo
框架更新	定期推送新功能	完全没有，靠团队开发
重构成本	完全不需要重构	基本整个框架需要整理一遍，大部分地方需要重构
最少研发费用	非常低，只需要一个月的团队支出	至少3-5个月的团队支出
定制服务	完全按照企业需求开发	完全没有，靠团队开发
法律责任	完全自主拥有知识产权	可能会被某些企业诉讼

NFrame

敏捷服务器开发解决方案

Q & A

谢谢

NFrame

敏捷服务器开发解决方案